Project2——Unix Shell 和历史特点

5142029014 高超

一、实验内容

此项目由修改一个 c 程序组成，它作为接收用户命令并在单独的进程执行每个命令的 Shell 接口。能接收一些简单的命令，要求实现后台运行，并创建历史特性。

二、实验环境

虚拟机上安装 Ubuntu 15.04 64 位。

三、实验的具体实现

本次实验可以分成两个大的内容，就是读入命令和执行命令。

先来看读命令，即 setup 函数，将读进来的字符数组 inputBuffe[]以空格为分隔符处理后放置于字符串数组 agrs[]中便于后面命令的执行。

1. 考虑到 r type 的命令实质上是执行历史命令中的某一条命令，同时 r type 命令在历史中是显示其指向的那条指令，而不是 r type 命令本身，于是先判断是否是 r type 的命令，从历史命令中寻找对应的命令赋给 inputBuffer[]。

```
//first deal with the r x, find the corresponding command from the history, and put it to the inputBuffer
if(inputBuffer[0] == 'r' && NumOfCommand > 0)
{
    if(inputBuffer[1] == '\n')
    {
        strcpy(inputBuffer, history[NumOfCommand - 1]);
    }
    else if(inputBuffer[1] == ' ' && inputBuffer[2] != '\0' && inputBuffer[3] == '\n')
    {
        for(i = NumOfCommand - 1; i >= 0; --i)
        {
            if(inputBuffer[2] == history[i][0])
            {
                strcpy(inputBuffer, history[i]);
                flag = 1;
                break;
            }
        }
        if(!flag)    //if there is no command match to the r x, then return and no action to the history
        {
            args[0] = NULL;
            return;
        }
```

2. 将 inputBuffer[]中的命令置于历史命令 history[]中。

```
if(NumOfCommand == 10) //if the num of history command is 10,move the olddest command out
{
    for(i = 0; i < 9; ++i)
    {
        strcpy(history[i], history[i + 1]);
    }
    strcpy(history[9], inputBuffer);
}
else
{
    strcpy(history[NumOfCommand], inputBuffer); //put it to the history
    ++NumOfCommand;
}
```

3. 将 inputBuffer[]命令分片放在 args[]中。

```
for(i = 0; i < len; ++i)
{
    if(inputBuffer[i] == ' ' || inputBuffer[i] == '\t') //find the delimiters
    {
        if(start != -1)
        {
            args[j++] = &inputBuffer[start];
            start = -1;
        }
        inputBuffer[i] = '\0';
    }
    else if(inputBuffer[i] == '\n')
    {
        if(start != -1)
        {
            //inputBuffer[i] = '\0';
            args[j++] = &inputBuffer[start];
        }
        args[j] = NULL;
        inputBuffer[i] = '\0';
    }
    else
    {
        if(inputBuffer[i] == '&')
        {
            *background = 1;
            inputBuffer[i] = '\0';
        }
        if(start == -1)
            start = i;
    }
}
```

再来看执行命令。

1. 参考书中的指导，使用系统调用函数 fork()产生一个子进程，使用 execvp 函数执行读入的命令。

```
if(args[0] != NULL)
{

    pid_t pid;

    pid = fork();

    if(pid < 0)
    {
        printf("fork failed\n");
        exit(1);
    }
    else if(pid == 0)
    {
        execvp(args[0], args);
        background = 0;
        printf("command error\n");
        exit(0);
    }

    else
    {
        if(background == 0)
        {
            waitpid(pid, NULL, 0);
        }
    }
```

2. 关于信号处理。

使用书中给出的信号处理函数 sigaction。

```
//set up the signal handler
struct sigaction handler;
handler.sa_handler = handle_SIGINT;
sigaction(SIGINT, &handler, NULL);
```

接收 Ctrl C 以后，打印出历史命令。

3. 关于后台执行。

使用 background 这个量来标记是否命令中有'&'，由此来判断父进程是否需要等待子进程的退出。

四、实验结果的呈现

一些简单常用的命令。

```
COMMAND-> pwd
/home/gao
COMMAND-> ps
  PID TTY          TIME CMD
10271 pts/3    00:00:00 bash
10292 pts/3    00:00:00 a.out
10319 pts/3    00:00:00 a.out
10321 pts/3    00:00:00 ps
COMMAND-> cal
     June 2016
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30

COMMAND-> ls
a.out              Music                    test.c
Desktop            Pictures                 Videos
Documents          producer_and_consumer.c  VMwareTools-9.6.2-1688356.tar.gz
Downloads          producer_and_consumer.c~ vmware-tools-distrib
```

r type 的命令

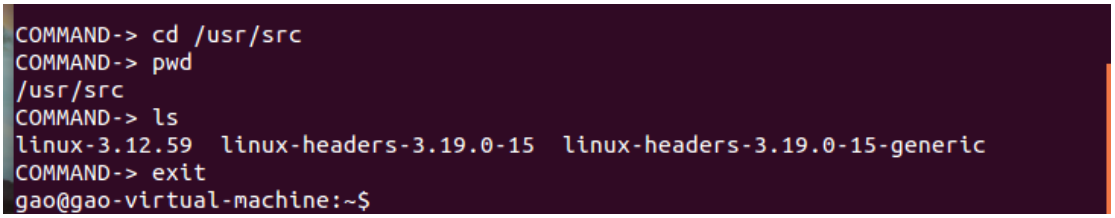
```
COMMAND-> date
Wed Jun  1 18:29:42 CST 2016
COMMAND-> r
Wed Jun  1 18:29:46 CST 2016
COMMAND-> r c
     June 2016
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30

COMMAND->
```

接收 Ctrl C 打印历史命令

```
COMMAND-> ^C
Caught Control C
The Command History:
1. pwd
2. ps
3. cal
4. ls
5. date
6. date
7. cal
COMMAND->
```

后台执行'&'

内建命令 cd 和 exit



五、实验的思考与讨论

1. 一开始屏幕上始终不打印出“command->”，后来发现是输出缓冲的问题，使用 fflush 函数强制刷新缓冲区。

2. 关于 r type 命令在何处处理，可以考虑在 main 函数中进行处理，但是由于每读入一个命令我就在 setup 函数中将其放入到历史命令中，而 r type 命令并不进入历史列表，它所指定的实际命令会进入，所以在 setup 函数最开始判断是否是 r type 命令。

3. 在后台运行中 wait 函数是有问题的，它是等待所有子进程的结束，所以改用 waitpid 函数，只等待我们创建的子进程的结束。

4. 事实上如此写完的 shell 是一个功能很不全的 shell，只能接收很少一部分命

令，当我输入 cd 命令时显示 error，在查阅一些资料后才知道，使用新的进程执行的命令是外部命令，而诸如 cd，exit 等命令是内建命令，是 shell 本身执行的命令，所以不能用 execvp 函数，需要自己写函数实现，关于 cd 命令使用系统调用函数 chdir 来实现。

```
//deal with cd command
if(strcmp(input, "cd") == 0)
{
    if(*args[1] == '~')
    {
        strcpy(args[1], "/home/gao");
    }
    if(chdir(args[1]) == 0)
    {
        continue;
    }
}
```

而 exit 直接使程序退出即可

```
if(strcmp(inputBuffer, "quit\n") == 0 || strcmp(inputBuffer, "exit\n") == 0)
    exit(0);
```

只实现以上两条内建命令。

六、实验总结

第二个 project 可以说是问题出现比较多，并且是比较复杂的一个程序。整体结构理清以后小问题一直涌现，关于字符串处理，信号的处理，创建子进程方面都有一些问题，所幸最后基本解决了问题，也学习到了许多。只是这个 project 有很多可以充实的，仍有待提高。